Práctica 1

Introducción a las Redes Neuronales Artificiales

3 de Marzo de 2014

AUTORES

Mauricio Guerreiro Quatrin
Olympia Muruzábal Ishigetani

Índice

Neuronas de McCulloch-Pitts

En este ejercicio se nos pide diseñar una red con neuronas de McCulloch-Pitts, con tres neuronas en la capa de entrada (x) y dos en la capa de salida (y). Esta red ha de distinguir la orientación del estimulo en los instantes t-1 y t-2, si es hacia arriba la salida en el instante t será (0 1), si es hacia abajo la salida será (1 0) y si está quieto será (0 0).

Como consideraciones adicionales, damos por supuesto que en cada instante de tiempo no habrá más de una entrada activa. Puesto que con más de una entrada activa no sería posible observar la orientación de los estímulos de entrada.

Diseño

Para resolver el problema planteado en este ejercicio primero obtenemos la siguiente expresión lógica para cada salida, según de la interpretación del enunciado. A partir de dicha expresión realizamos el posterior diseño de la red.

$$Y_1(t) = \big(x_1(t-2) and x_2(t-1)\big) or \big(x_2(t-2) and x_3(t-1)\big) or \big(x_3(t-2) and x_1(t-1)\big)$$
$$= (x_1(t-2)\ and\ z_2) or (x_2(t-2) and\ z_3) or (x_3(t-2) and\ z_1)$$
$$= A_1 or\ A_2 or\ A_3$$

$$Y_2(t) = \big(x_1(t-2) and x_3(t-1)\big) or \big(x_3(t-2) and x_2(t-1)\big) or \big(x_2(t-2) and x_1(t-1)\big)$$
$$= (x_1(t-2)\ and\ z_3) or (x_3(t-2) and\ z_2) or (x_2(t-2) and\ z_1)$$
$$= B_1 or\ B_2 or\ B_3$$

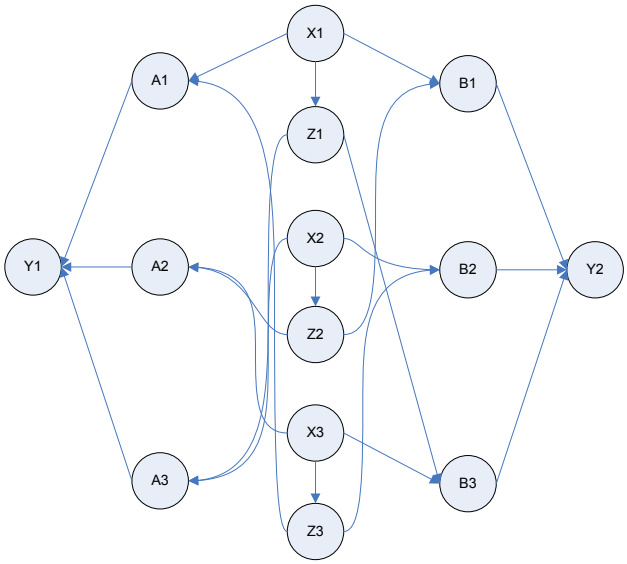


Figura 1 Diseño de la red de McCulloch-Pitts

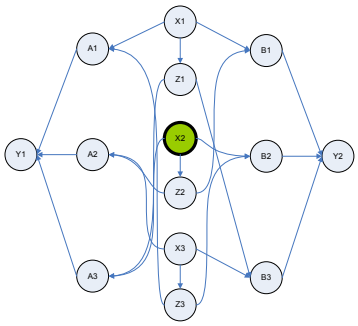
Siendo el umbral de las neuronas de la red 2 el peso de las entradas será 1 en el caso de las operaciones lógicas and y 2 en el caso del or. Dado el diseño podemos predecir que serán necesarios tres tiempos hasta poder predecir la primera entrada.

Resultados

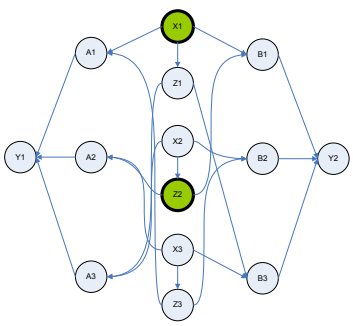
A continuación, se muestran en varios ejemplos la activación de todas las neuronas en cada unidad de tiempo. Los nodos marcados están activos, es decir adquieren un valor de 1, mientras que los que no están marcados tienen un valor de 0.

Arriba (0 1):

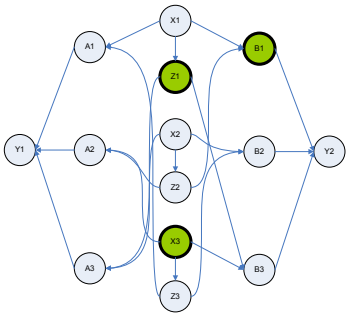
t=0



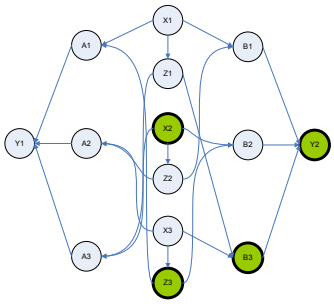
t=1



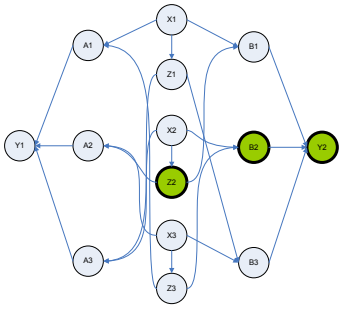
t=2



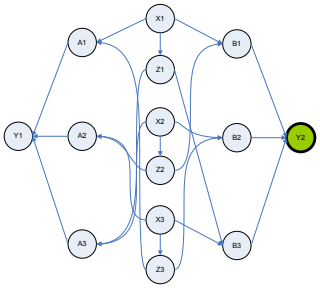
t=3



t=4

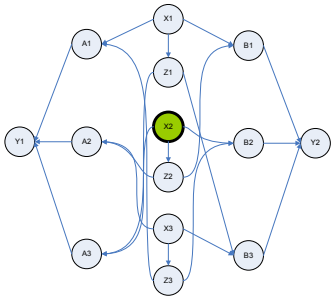


t=5

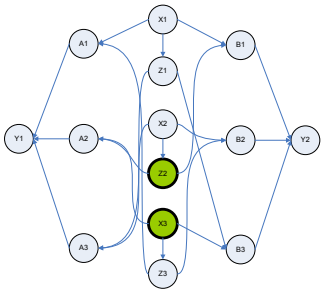


Abajo (1 0):

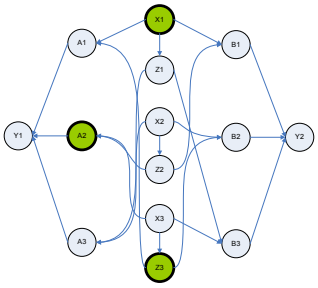
t=0



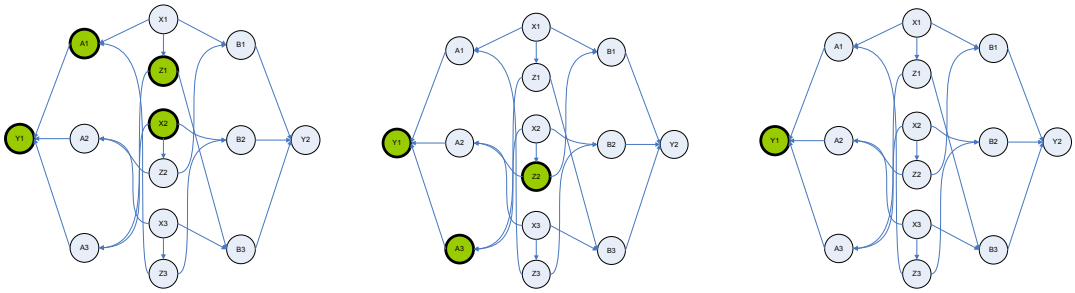
t=1



t=2



t=3 t=4 t=5



Para ejecutar el código hay que pasar como parámetros de entrada la ruta al fichero de entrada y el fichero de salida, donde se guardarán los estados de cada neurona en cada paso de tiempo. En los ficheros adjuntos resultsUp.txt, resultsDown.txt y resultsData.txt se pueden observar los estados de cada neurona en cada instante de tiempo de las pruebas realizadas.

Perceptrón y Adaline

En este apartado realizaremos un estudio sobre las reglas de aprendizaje del Perceptrón y Adaline y su aplicación a la clasificación de patrones. Para la implementación de las dos redes nos basamos en la teoría dada en clase.

Perceptrón

En este caso implementamos un perceptrón simple puesto que no tiene capas ocultas, solo una de entrada y otra de salida. Que a diferencia del perceptrón multicapa solo es capaz de resolver problemas linealmente separables.

Diseño

Antes de describir la clase ***Perceptron***, vamos a hablar un poco sobre las otras clases auxiliares.

- ***FileService***
  - Servicio sin estado que solamente hace la lectura (**read**) y grabación (**save**) de ficheros.
- ***InputData***
  - Es responsable de preparar los datos leídos por el ***FileService***. Debe interpretar las líneas del fichero (**prepareRows**), generar datos exactamente del mismo padrón de entrada, pero con las líneas entrelazadas de manera aleatoria (**shuffleFileLines**) y repartir los datos a partir de un porcentaje solicitado (**getData**).
- ***InputRow***
  - Representa los datos de una línea del fichero de entrada. Su objetivo es hacer la gestión de los valores de cada neurona de entrada y su respectiva clase de salida (**getTargetRepresentation**).
- ***OutputData***

- Esta clase trabaja en el control del fichero de salida. Perceptron y Adaline ambos utilizan como contenedor del procesamiento a cada ***InputRow*** leída.

- ***Perceptron***
  - Su funcionamiento se basa en:
    - **setData:** prepara los datos de entrada con ayuda de ***InputData***.
    - **initializeWeights:** inicializa el array de pesos y el sesgo en cero.
    - **calculatePartialResponse:** hace el calculo de **y_in** respeto a una ***InputRow*.**
    - **calculateResponse:** basado en la salida de **calculatePartialResponse** y la codificación de la neurona (bipolar), calcula la **f (y_in)** de la neurona.
    - **updateWeights:** dada una ***InputRow*** y su *target* correspondiente, actualiza los pesos de cada neurona de entrada.
    - **startTraining:** ejecuta el entrenamiento de la neurona como visto en teoría para cada ***InputRow*** del ***InputData*** respectivo, llamando en secuencia **calculatePartialResponse, calculateResponse** y si hay error llamando al **updateWeights.**
    - **startTest:** ejecuta el test de la neurona como visto en teoría, llamando en secuencia **calculatePartialResponse** y **calculateResponse.** Con la ayuda de ***OutputData***, graba los resultados a cada ***InputRow*** procesado.

Resultados

Tras programar el perceptrón, hemos comprobado su eficacia frente a varias fuentes de datos, cambiando los distintos parámetros de la red, como el número máximo de épocas o la constante de aprendizaje.

Primero con patrones linealmente separables que conocemos, como la operación lógica nand, cuyas tasas de error eran 0% tal como se esperaba. Después probamos con patrones que no son separables linealmente, como xor, cuya tasa de error variaba pero nunca era del 0%.

Posteriormente realizamos pruebas con las bases de datos que se nos proporcionaban prueba_real1.txt y prueba_real2.txt. Dado que las tasas de error eran mayores, hemos podido deducir que los datos de ambos ficheros no son linealmente separables.

Adaline

Adaline y el perceptrón tienen básicamente el mismo algoritmo de aprendizaje, la diferencia reside en la regla de aprendizaje, conocida como regla delta o de mínimo error cuadrático medio.

Diseño

Ahora vamos a describir la clase ***Adaline***. Las clases auxiliares son las mismas ya documentadas en el apartado anterior.

- ***Adaline***
  Su funcionamiento se basa en:
- **setData:** prepara los datos de entrada con ayuda de ***InputData***.

- **initializeWeights:** inicializa el array de pesos y el sesgo con valores aleatorios entre [-0.5;0.5].
- **calculatePartialResponse:** hace el cálculo de **y_in** respeto a una ***InputRow*.**
- **calculateResponse:** basado en la salida de **calculatePartialResponse** y la codificación de la neurona (bipolar), calcula la **f (y_in)** de la neurona. Aquí diferente del Perceptron no hay indecisión, la salida debe ser 1 o -1.
- **updateWeights:** dada una ***InputRow,*** su *target* correspondiente y la salida de **calculatePartialResponse,** actualiza los pesos de cada neurona de entrada y el sesgo con base en el delta calculado.
- **startTraining:** ejecuta el entrenamiento de la neurona como hemos visto en teoría para cada ***InputRow*** del ***InputData*** respectivo, llamando al **calculatePartialResponse** y al **updateWeights** para actualizar la condición de parada (delta).
- **startTest:** ejecuta el test de la neurona como lo visto en teoría, llamando en secuencia **calculatePartialResponse** y **calculateResponse.** Con la ayuda de ***OutputData***, graba los resultados a cada ***InputRow*** procesado.

Resultados

Realizamos las mismas pruebas que aplicamos al perceptrón con la diferencia de que ajustamos a su vez la condición de parada. Comprobamos que la condición de parada que utilizamos –el mayor cambio de peso- hace que varíe el resultado de la red. Cuanto mayor es el cambio de peso mayor es la precisión de la red, si la diferencia es muy pequeña es posible que la red no aprenda en el entrenamiento. Con los resultados obtenidos reafirmamos las observaciones encontradas con el perceptrón.

Discusión y Comparación de los Resultados

Tras las pruebas realizadas para ambas redes podemos proceder a realizar la comparativa. Los resultados del error promedio frente al número de épocas de las bases de datos problema_real1.txt y problema_real2.txt se muestran en el anexo. En la serie1 se pueden observar las muestras obtenidas y en las mismas gráficas su tendencia logarítmica. Salvo en casos excepcionales, la mayoría de las gráficas muestra una tendencia decreciente coherente con el entrenamiento. Por lo que hemos comprobado que la tasa de aprendizaje que más se ajusta es 0.3 y donde se estabiliza el aprendizaje es aproximadamente a las 16 épocas.

Ambas redes resuelven problemas linealmente separables, es decir pueden realizar una separación de clases mediante hiperplanos. Por lo que resuelven sin problemas las operaciones nand y nor, pero no es así con xor o las bases de datos problema_real1 y problema_real2 que no son linealmente separables.

Si añadimos entradas adicionales, como combinaciones no lineales de los atributos originales, es posible solucionar problemas como el xor, puesto que cambiamos las dimensiones del problema lo que hace posible separar las clases con hiperplanos. Pero haciendo varias pruebas con las bases de datos reales comprobamos que nos siempre se mejora la predicción, depende de las entradas añadidas.

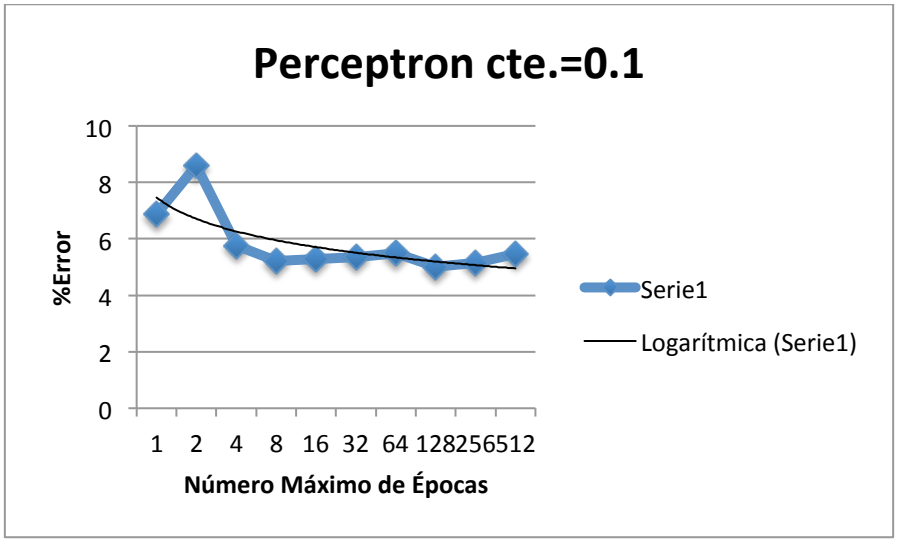
Hemos de tomar algunas precauciones para que una red neuronal aprenda adecuadamente a generalizar y no sobreajuste ya que, si sobreaprende algunos casos, los resultados en el análisis de algunas muestras pueden ser pésimos. El sobreajuste puede darse por un número excesivo de ciclos de aprendizaje o un número reducido de patrones de entrenamiento. Una medida es un correcto entrenamiento, desordenando adecuadamente el conjunto de datos. Tras aplicar esto por primera vez en la implementación de nuestros clasificadores, los resultados han sido mucho mejores, comprobando de esta forma que realmente funcionan.

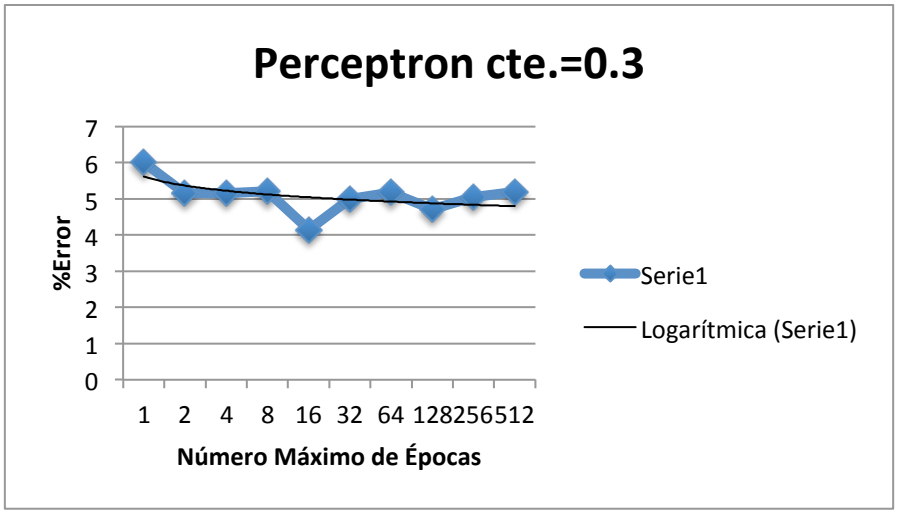
Cabe mencionar que el método del gradiente que utilizamos como algoritmo de aprendizaje nos puede llevar a un mínimo local. Para evitar esto podemos partir de inicializaciones aleatorias.

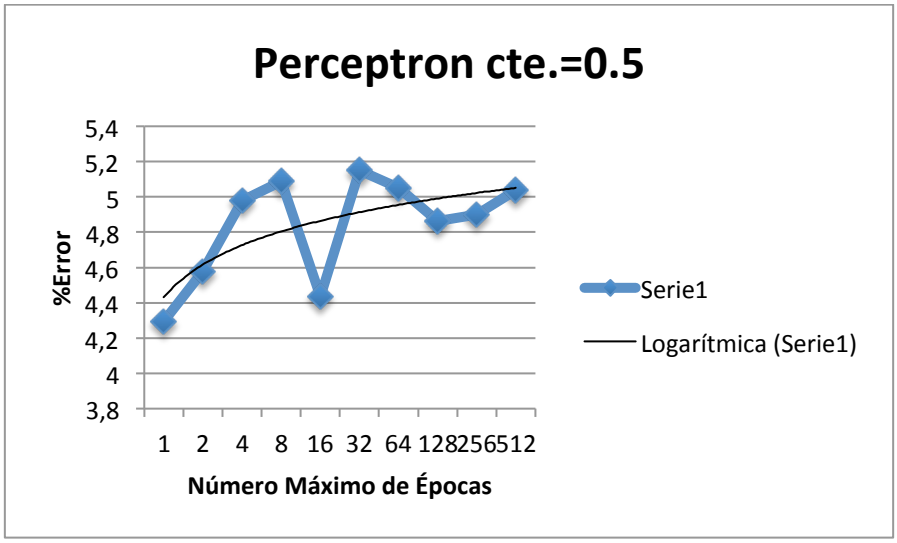
Por último, para llegar a una tasa de fallos mínima, ha sido necesario realizar un ajuste muy fino sobre alguno de los parámetros, como es la constante de aprendizaje.

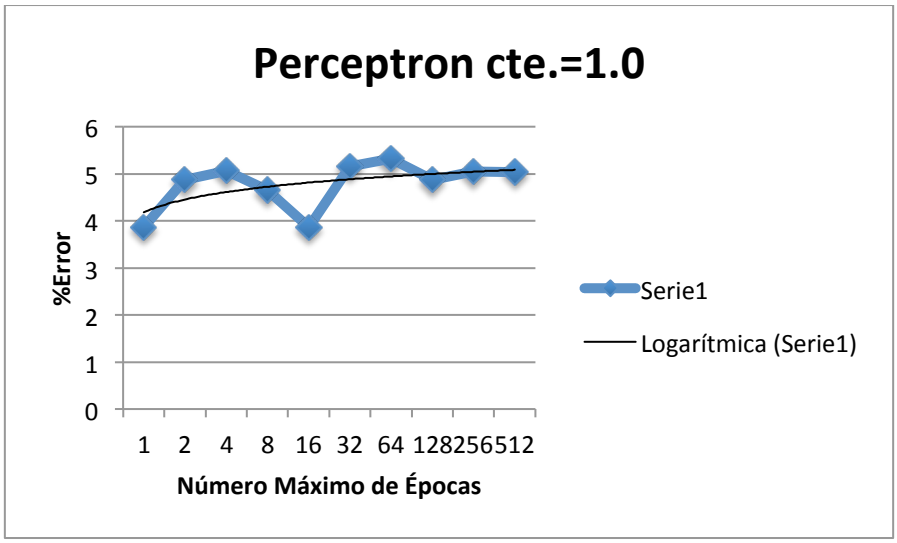
Anexos

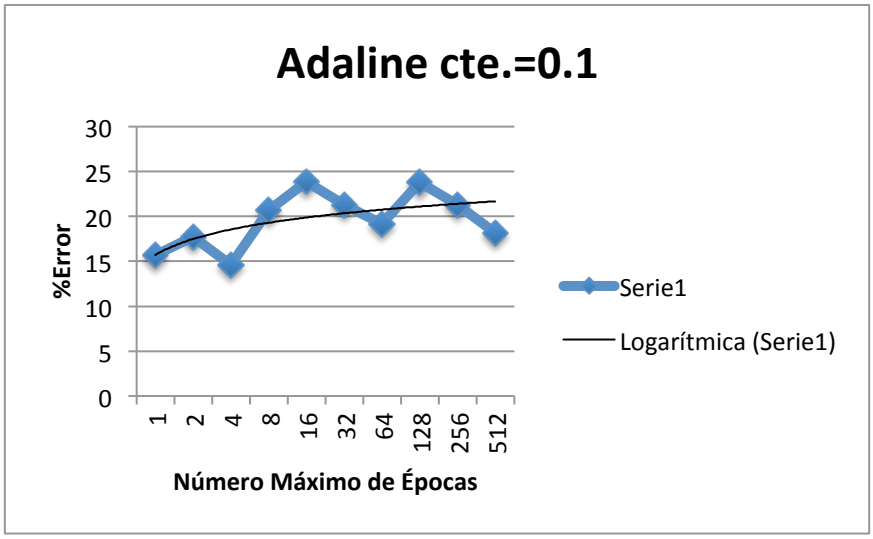
Resultados del problema_real1.txt

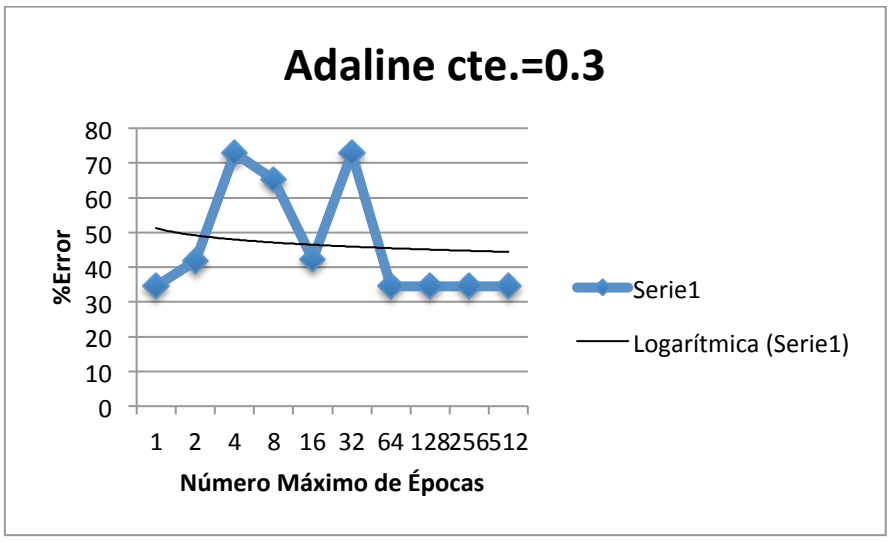


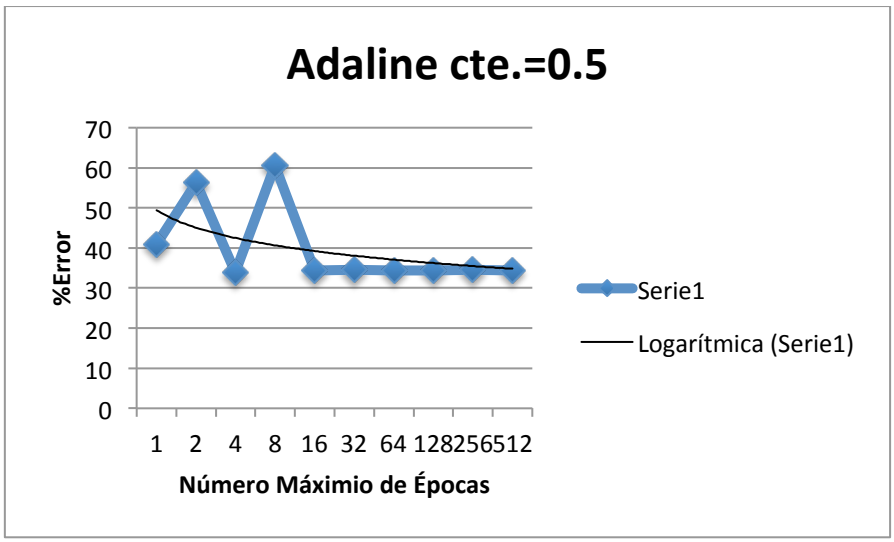

Perceptron cte.=0.3
%Error
7
6
5
4
3
2
1
0
1 2 4 8 16 32 64 128256512
Número Máximo de Épocas
Serie1
Logarítmica (Serie1)

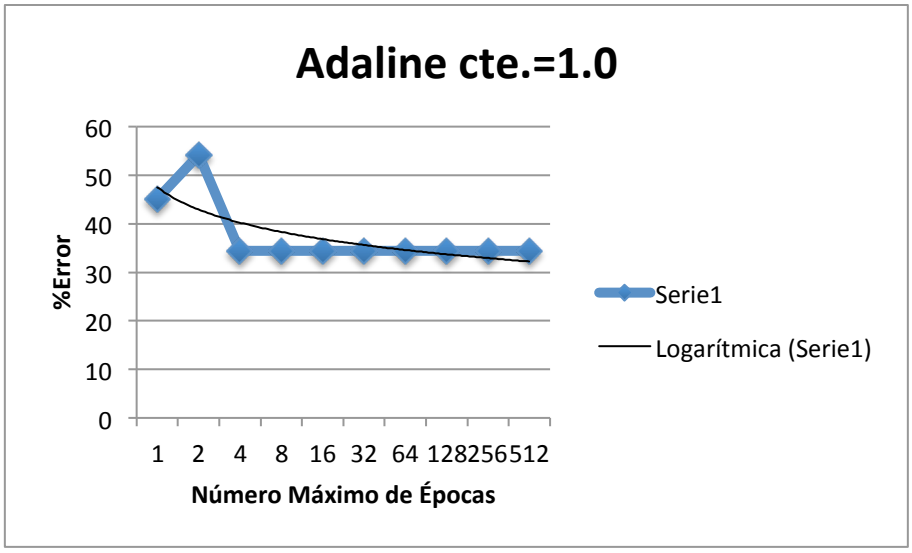

Perceptron cte.=0.5
%Error
5,4
5,2
5
4,8
4,6
4,4
4,2
4
3,8
1 2 4 8 16 32 64 128256512
Número Máximo de Épocas
Serie1
Logarítmica (Serie1)


Perceptron cte.=1.0
%Error
6
5
4
3
2
1
0
1 2 4 8 16 32 64 128256512
Número Máximo de Épocas
Serie1
Logarítmica (Serie1)

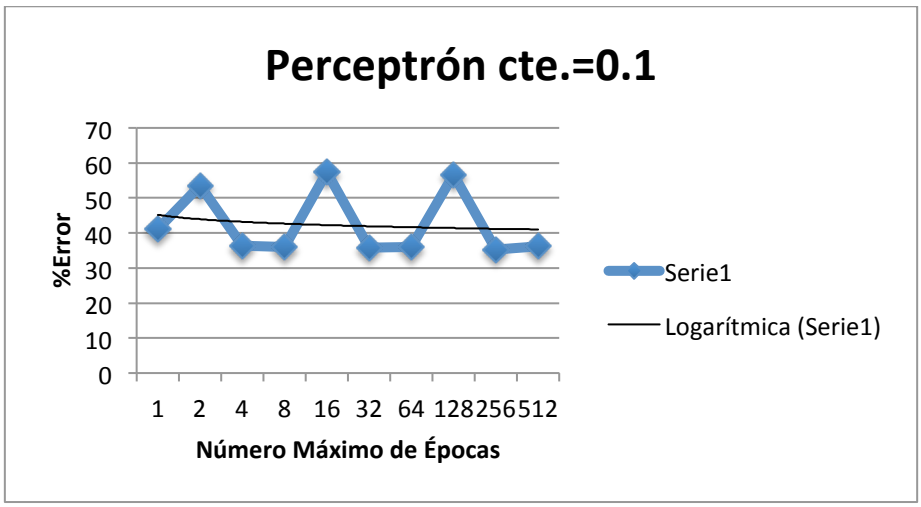

Adaline cte.=0.1
%Error
30
25
20
15
10
5
0
1
2
4
8
16
32
64
128
256
512
Número Máximo de Épocas
Serie1
Logarítmica (Serie1)

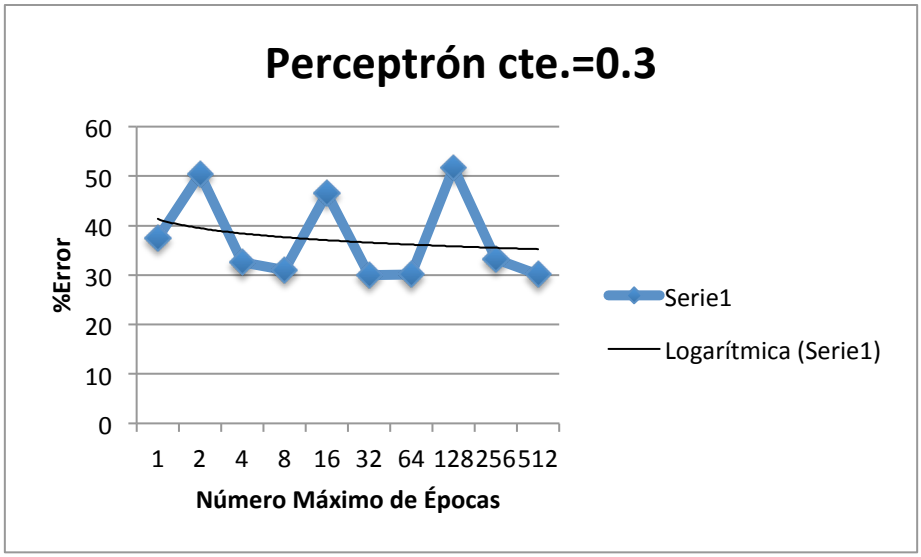

Adaline cte.=0.3
%Error
80
70
60
50
40
30
20
10
0
1 2 4 8 16 32 64 128256512
Número Máximo de Épocas
Serie1
Logarítmica (Serie1)

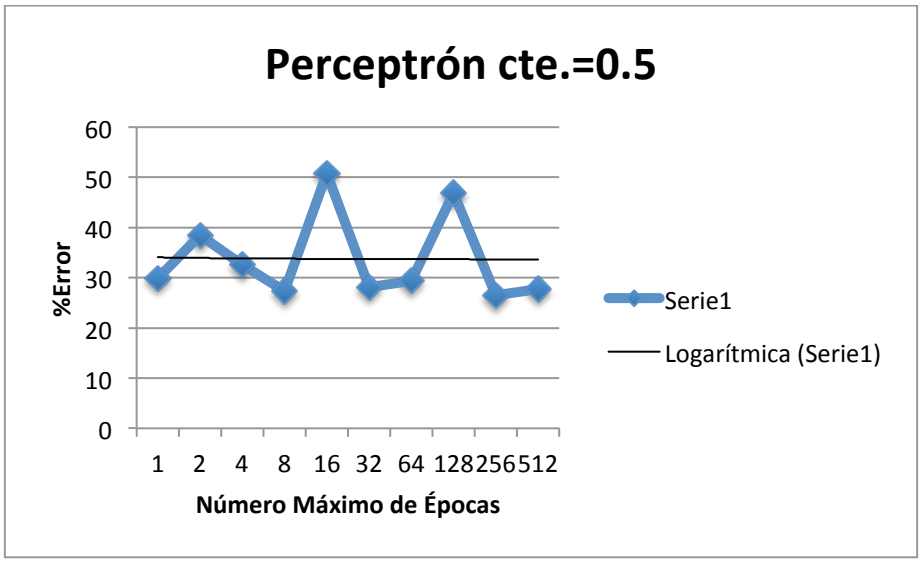

Adaline cte.=0.5
%Error
70
60
50
40
30
20
10
0
1 2 4 8 16 32 64 128256512
Número Máximio de Épocas
Serie1
Logarítmica (Serie1)

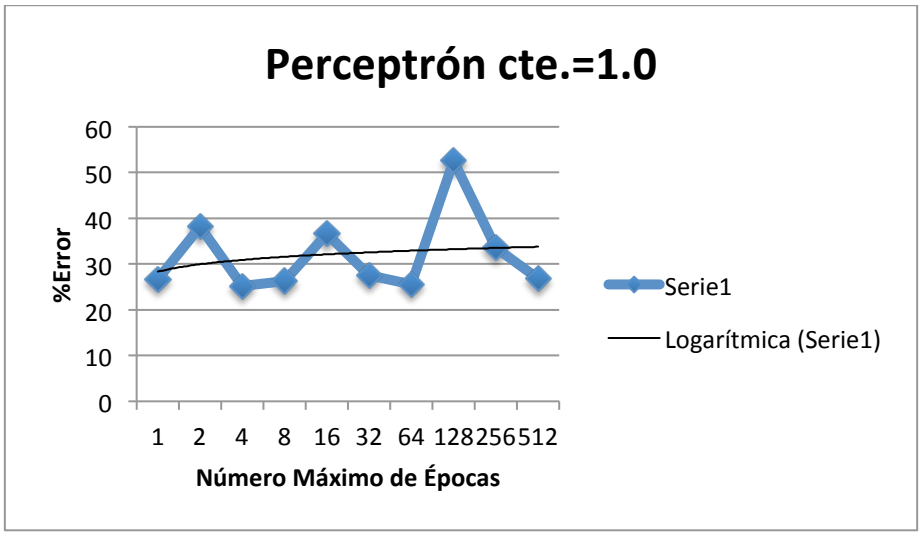


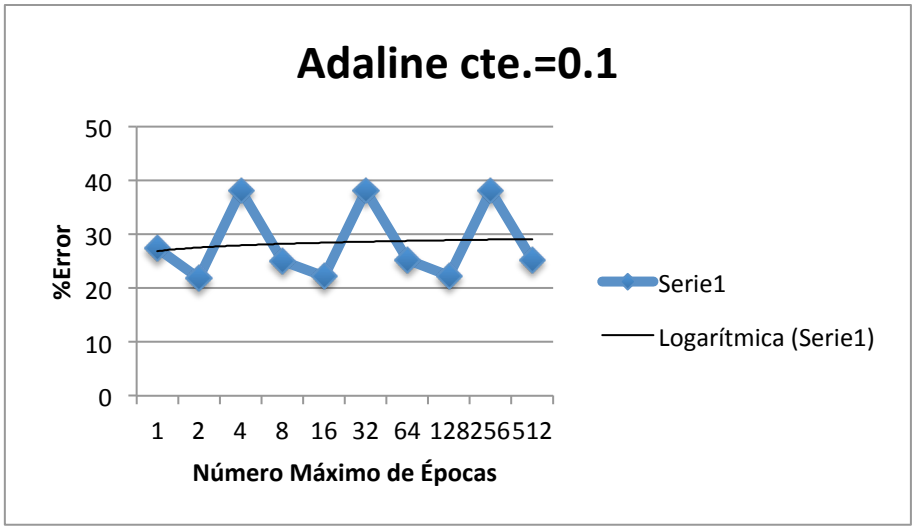
Resultados del problema_real2.txt

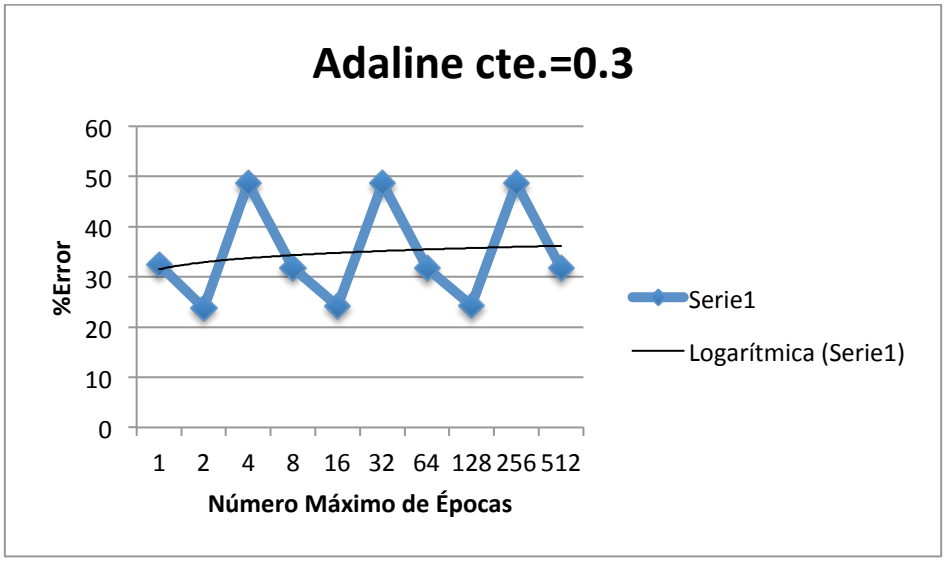


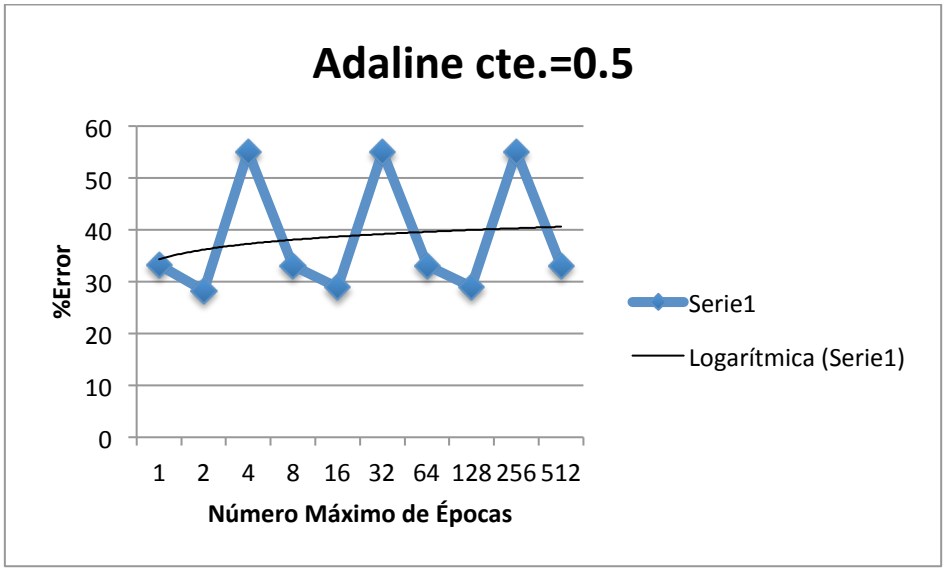

Perceptrón cte.=0.3
%Error
60
50
40
30
20
10
0
1 2 4 8 16 32 64 128256512
Número Máximo de Épocas
Serie1
Logarítmica (Serie1)

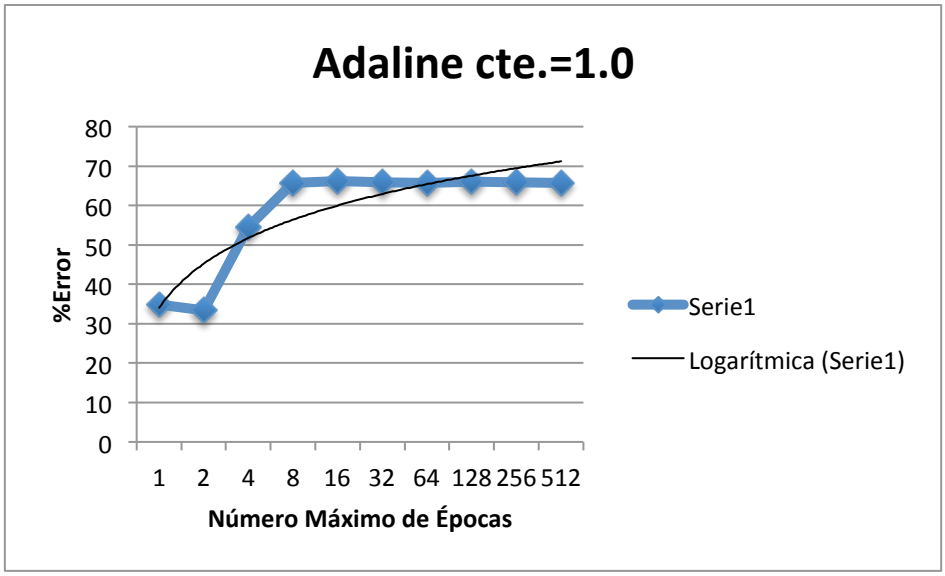

Perceptrón cte.=0.5
%Error
60
50
40
30
20
10
0
1 2 4 8 16 32 64 128256512
Número Máximo de Épocas
Serie1
Logarítmica (Serie1)


Perceptrón cte.=1.0
%Error
60
50
40
30
20
10
0
1 2 4 8 16 32 64 128256512
Número Máximo de Épocas
Serie1
Logarítmica (Serie1)


Adaline cte.=0.1
%Error
50
40
30
20
10
0
1 2 4 8 16 32 64 128256512
Número Máximo de Épocas
Serie1
Logarítmica (Serie1)


Adaline cte.=0.3
%Error
60
50
40
30
20
10
0
1 2 4 8 16 32 64 128 256 512
Número Máximo de Épocas
Serie1
Logarítmica (Serie1)


Adaline cte.=0.5
%Error
60
50
40
30
20
10
0
1 2 4 8 16 32 64 128 256 512
Número Máximo de Épocas
Serie1
Logarítmica (Serie1)


Adaline cte.=1.0
80
70
60
50
40
30
20
10
0
%Error
1
2
4
8
16
32
64
128
256
512
Número Máximo de Épocas
Serie1
Logarítmica (Serie1)